International Journal of Foundations of Computer Science

Computing the Expected Edit Distance from a String to a Probabilistic Finite-State Automaton

Jorge Calvo-Zaragoza, Jose Oncina

Department of Software and Computing Systems
University of Alicante, Carretera de San Vicente s/n, Alicante, Spain
{*jcalvo,oncina*}*@dlsi.ua.es*

Colin de la Higuera

Laboratoire d'Informatique de Nantes-Atlantique
University of Nantes, UMR 6241, Nantes, France
cdlh@univ-nantes.fr

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

In a number of fields, it is necessary to compare a witness string with a distribution. One possibility is to compute the probability of the string for that distribution. Another, giving a more global view, is to compute the expected edit distance from a string randomly drawn to the witness string. This number is often used to measure the performance of a prediction, the goal then being to return the *median* string, or the string with smallest expected distance. To be able to measure this, computing the distance between a hypothesis and that distribution is necessary. This paper proposes two solutions for computing this value, when the distribution is defined with a probabilistic finite state automaton. The first is exact but has a cost which can be exponential in the length of the input string, whereas the second is a fully polynomial-time randomized schema.

Keywords: Edit distance; Probabilistic Finite State Automata, Median string.

1. Introduction

The edit or Levenshtein distance is often used to measure how close one string is to another [14]. This distance has given rise to many questions: if one is given a set instead of a string, the question may be to compute rapidly the distance between the set and a string or between two sets [29, 17]. In turn, a set defines an empirical distribution which can be represented by a probabilistic finite state automaton (Pfa), a hidden Markov model or a weighted automaton [17, 28].

If the set is used as a learning sample, the distribution may be more general, again represented by the above machines, but these may now contain cycles and therefore define a distribution over all possible strings.

The following questions are then posed: what is the expected distance between a given *witness* string and such a distribution? What could a representative string

be for this distribution? One possible answer to the second question is the *most probable string* [8, 9]. Another possible answer is the *median string*, which is the string minimizing the expected distance to the distribution. This option, in turn, contributes to make the first question relevant. Note that the median can be a null-probability string.

These questions have not only a precise mathematical interest, but they have been posed in very different settings like bio-informatics [10], pattern recognition [19] or computational linguistics [24].

Alternative distances have been studied, such as the minimum cost obtaining by summing the weight of a string and its distance to the witness string [2]. Balls of strings, Levenshtein automata and other finite state machines linking regular languages and the edit distance have been introduced, discussed and studied [17, 4, 16, 23].

In the conference version of this paper [6] we presented two results. The first is that the expected edit distance is computable, and that when the weights of the PFA are rational, then the result itself is rational as well. The construction involves building a multiplicity automaton which can be of size exponential in the length of the string w, but only increases polynomially with the number of states of the PFA or the size of the alphabet. The second result is that the problem admits a *fully polynomial time randomized schema* (FPRAS), that is, a randomized algorithm which will return a probably approximatively correct value.

In this extended version, we provide a more comprehensive construction of the multiplicity automaton, including a step-by-step example. We also include the demonstration of the polynomial bound with respect to the length of the string, the size of the automaton representing the distribution, and the inverse of the accepted error of the proposed FPRAS.

As we will show in the experimental section, the method involving the multiplicity automaton will give an exact result, but only for small witness strings. The FPRAS, on the other hand, can only build an approximate result, but there is a guarantee on the error bound and the method can handle long witness strings and large PFA.

After introducing notations and definitions (Section 2), we prove in Section 3 that the problem is decidable and provide an algorithm which gives the correct result; Section 4 presents a polynomial randomized computation whose result is probably approximately correct. Our experiments, described in Section 5, empirically confirm the bounds in both error and complexity of the proposed strategies. Section 6 concludes the present work.

2. Preliminaries

2.1. *Basic Notations*

An *alphabet* Σ is a finite non-empty set of symbols called *letters*. A *string* w over Σ is a finite sequence $w = w_1 \dots w_m$ of letters. Letters will be indicated by $a, b, c, \dots$,

and strings by $u, v, \ldots, z$. Let $|w|$ denote the length of w. In this case we have $|w| = |w_1 \ldots w_m| = m$. The *empty string* is denoted by λ.

We denote by Σ^* the set of all strings, by Σ^m the set of those of length m.

A *probabilistic language* $\mathcal{D}$ is a probability distribution over Σ^*. The probability of a string $w \in \Sigma^*$ under the distribution $\mathcal{D}$ is denoted as $\Pr_{\mathcal{D}}(w)$. The distribution must satisfy $\sum_{w \in \Sigma^*} \Pr_{\mathcal{D}}(w) = 1$.

If the distribution is modelled by some syntactic machine $\mathcal{M}$, the probability of x according to the probability distribution defined by $\mathcal{M}$ is denoted by $\Pr_{x \sim \mathcal{M}}(x)$ or simply $\Pr_{\mathcal{M}}(x)$.

2.2. *Multiplicity Automata*

An n-state Multiplicity Automaton (MA) $\mathcal{M}$ (also known as *recognizable series* [5] or *Stochastic Sequential Machines* [20]) can be defined by a 4-tuple $\langle \Sigma, \mathbf{S}, \mathbf{M}, \mathbf{F} \rangle$ where: Σ is the alphabet, $\mathbf{S} \in \mathbb{Q}^{1 \times n}$, $\mathbf{M} = \{\mathbf{M}_a \in \mathbb{Q}^{n \times n} : a \in \Sigma\}$, and $\mathbf{F} \in \mathbb{Q}^{n \times 1}$.

$\mathcal{M}$ realizes a function from Σ^* to $\mathbb{Q}$ such that:

$$\mathcal{M}(x_1 \cdots x_k) = \mathbf{S} \sum_{i=1}^{k} \mathbf{M}_{x_i} \mathbf{F}.$$

This machine can also be defined from a graph point of view as an n-state machine $\langle \Sigma, Q, \mathbb{S}, \mathbb{F}, \delta \rangle$ where $Q = \{q_0, \cdots, q_{n-1}\}$, $\mathbb{S} : Q \to \mathbb{Q}$ is the function that assigns an initial weight to each state ($\mathbb{S}(q_i) = \mathbf{S}[i]$), $\mathbb{F} : Q \to \mathbb{Q}$ is the function assigning a final weight to each state ($\mathbb{F}(q_i) = \mathbf{F}[i]$), and $\delta : Q \times \Sigma \times Q \to \mathbb{Q}$ is the function that assigns a probability to each transition ($\delta(q_i, a, q_j) = [\mathbf{M}_a]_{i,j}$).

Given $x \in \Sigma^*$, $\Pi_{\mathcal{M}}(x)$ is the set of all paths accepting x: an *accepting* x-path is a sequence $\pi = q_{i_0} x_1 q_{i_1} x_2 \ldots x_k q_{i_k}$ where $x = x_1 \cdots x_k$, $x_i \in \Sigma$, and $\forall j \in [1, k]$ such that $\delta(q_{i_{j-1}}, x_j, q_{i_j}) \neq 0$. Let $\pi = q_{i_0} x_1 q_{i_1} x_2 \ldots x_k q_{i_k}$, we denote by $\delta(\pi) = \prod_{j=1}^{k} \delta(q_{i_{j-1}}, x_j, q_{i_j})$, $\alpha(\pi) = q_{i_0}$ and $\omega(\pi) = q_{i_k}$. In this case, the function $\mathcal{M}$ is defined as:

$$\mathcal{M}(x) = \sum_{\pi \in \Pi_{\mathcal{M}}(x)} \mathbb{S}(\alpha(\pi)) \delta(\pi) \mathbb{F}(\omega(\pi)),$$

which can be computed using the Forward (or Backwards) algorithm [27] in $O(|x|\,|Q|^2)$. Obviously, the two ways to compute $\mathcal{M}(x)$ are equivalent.

Probabilistic Finite Automata (PFA) can be viewed as a special type of MA that are restricted to describe probability distributions over sets of strings. Then further restrictions should be applied. Let $\mathbf{1} \in \mathbb{Q}^{n \times 1} : \mathbf{1}[i] = 1 \forall i$, $I \in \mathbb{Q}^{n \times n}$ be the identity matrix and $\mathbf{M}_\Sigma = \sum_{a \in \Sigma} \mathbf{M}_a$, then:

- the components of $\mathbf{S}$, $\mathbf{M}$ and $\mathbf{F}$ are interpreted as probabilities, that is, they should be in $[0, 1]$.
- $\mathbf{S1} = 1$: the sum of the starting probabilities is one.

- $\mathbf{M}_\Sigma \mathbf{1} + \mathbf{F} = \mathbf{1}$: for any state, the sum of the outgoing probability plus the ending probability is one.
- $(I - \mathbf{M}_\Sigma)$ should be non-singular: this is a sufficient condition to assure the non existence of sink states (states from which the probability of any string is null).

2.3. *The Edit Distance*

The edit distance between two strings $d_e(w, u)$ is the minimum number of edit operations needed to transform w into u [14].

We will make use of the following generous bounds for the edit distance:

$$d_e(w, u) \leq \max\{|w|, |u|\} \leq |w| + |u| \tag{1}$$

The *relative edit distance* from w to u is $d_r(w, u) = d_e(w, u) - |u|$. Notice that this is not a metric.

It follows from (1) that for a fixed string w the set of values that $d_r(w, u)$ can take is finite, with values ranging from $-|w|$ to $|w|$, even though the set of strings from which u is chosen is infinite.

We extend the edit distance definition to distributions over strings (*string-distribution edit distance*):

$$d_e(w, \mathcal{D}) = \sum_{x \in \Sigma^*} d_e(w, x) \Pr_{\mathcal{D}}(x) = \sum_{x \in \Sigma^*} d_r(w, x) \Pr_{\mathcal{D}}(x) + \sum_{x \in \Sigma^*} |x| \Pr_{\mathcal{D}}(x) \tag{2}$$

When $\mathcal{D}$ is given by a PFA $\mathcal{A}$, we can also write $d_e(w, \mathcal{A})$.

2.4. *Complexity Issues*

Let us recall that a decision problem is one for which the possible answers are **true** and **false**. Such a problem is in class **P** if there is a deterministic Turing machine solving any instance in polynomial time, in **NP** if this machine is non-deterministic, **NP**-complete if it is in **NP** and it is as hard as any of the other **NP**-complete problems.

An optimization problem asks for a numerical value to be computed given an instance. This value can sometimes be approximated by a polynomial-time approximation scheme (PTAS) which can compute a value within a factor $1+\epsilon$ of the optimum in time polynomial in the size of the approximation scheme. If the runtime also depends polynomially of $1/\epsilon$, the scheme is called a fully polynomial-time approximation scheme or FPTAS. For more about approximation algorithms, see [26].

Sometimes, deterministic algorithms are unable to approximate, but randomized algorithms [18] can solve the problem in the following sense: an algorithm **A** is a *fully polynomial time randomized schema* or FPRAS if it can return a solution which is at distance ϵ of the optimum, with confidence at least $1 - \delta$ and runs in time polynomial in the size of the instance, $1/\epsilon$ and $1/\delta$.

The key problem in this work is called EDD:

Name: Computing the edit distance to a distribution generated by a PFA (EDD)
Instance: A PFA $\mathcal{A}$ over an alphabet Σ. A string w over Σ.
Question: Compute $d_e(w, \mathcal{A})$.

If we need to only consider the decision problem we will be also taking a rational input r and asking if $d_e(w, \mathcal{A}) \leq r$. And the associated approximation problem consists in computing a value t such $|t - d_e(w, \mathcal{D})| < \epsilon$.

The exact status of EDD is an open question. We conjecture it is **NP**-hard.

3. EDD is Decidable

We first prove that there exists an algorithm which takes a string w and a PFA $\mathcal{A}_\mathcal{D}$ and computes $d_e(w, \mathcal{A}_\mathcal{D})$. The computation cannot be bounded by a polynomial, but it terminates. The construction we propose follows three steps:

(1) We first (Sect. 3.1) build from w an MA $\mathcal{A}_w$ which can compute $d_r(w, u)$ for any u.
(2) We next (Sect. 3.2) build from $\mathcal{A}_\mathcal{D}$ and $\mathcal{A}_w$ an MA $\mathcal{A}_{\mathcal{D},w}$ which computes the product of the relative edit distance and the probability of the string.
(3) Using the matrix representation of $\mathcal{A}_{\mathcal{D},w}$ and $\mathcal{A}_\mathcal{D}$ we are able to compute the values of the infinite series $\sum_{x\in\Sigma^*} |x| \Pr_{\mathcal{A}_\mathcal{D}}(x)$ and $\sum_{x\in\Sigma^*} d_r(w, x) \Pr_{\mathcal{A}_\mathcal{D}}(x)$.

3.1. *Building a Multiplicity Automaton Computing the Edit Distance to a String (Step 1)*

Given a string w, we build (with Algorithm 1 MA_BUILD) an MA, $\mathcal{A}_w$, which will allow us to parse any other string u and, in linear time, obtain $d_r(w, u)$.

One way to implement the classic algorithm in order to only be linear in space is to compute one column at a time. The states of the MA are the different columns one may obtain when running the classical edit distance algorithm for strings w (used to index the lines) and u (used to index the columns), and substracting, in each cell, the length u, ie, computing $d_r(w, u)$, with w fixed and u being any string. For the sake of clarity, Tables 1 and 2 show the computation of the edit distance and the relative edit distance between $w =$ `ab` and $u =$ `baaaba`.

There is a transition in the MA labelled by symbol `a` between the state corresponding to the last column of $d_r(w, u)$ to the state corresponding to the last column of $d_r(w, ua)$, for some string u. Therefore, each state of the MA is denoted by a vector $[v_0, \ldots, v_m]$. The number of states is finite, because $d_r(w, \cdot) \in [-|w|, |w|]$, so the number of possible columns is bounded by $(2|w + 1|)^{|w+1|}$. Moreover, if we take into account that the difference between two consecutive elements in a column is in $\{-1, 0, 1\}$, the number of different columns, hence of states, is bounded by $3^{|w|}$.

There is no guarantee that the construction terminates in polynomial time. Yet

Algorithm MA_Build(w)
Data: $w = w_1 \dots w_m$ of length m
Result: a multiplicity automaton $\mathcal{A}_w = \langle \Sigma, Q, \mathbb{S}, \mathbb{F}, \delta \rangle$

$q_0 \leftarrow [0, 1, 2, \dots, m]$; $Q \leftarrow \{q_0\}$; $\mathbb{S}(q_0) \leftarrow 1$; $\mathbb{F}(q_0) \leftarrow m$;
unmarked $\leftarrow \{q_0\}$;
while unmarked $\neq \emptyset$ **do**
  Choose q in unmarked ;
  unmarked $\leftarrow$ unmarked $- \{q\}$;
  for $a \in \Sigma$ **do**
    $q'[0] \leftarrow 0$;
    for $i = 1$ **to** m **do**
      if $w_i = a$ **then** $x \leftarrow 0$;
      else $x \leftarrow 1$;
      $q'[i] \leftarrow \min\{q[i], q[i-1] + x - 1, q'[i-1]\}$;
    if $q' \in Q$ **then** $\delta(q, a, q') \leftarrow 1$;
    else
      $Q \leftarrow Q \cup \{q'\}$; $\delta(q, a, q') \leftarrow 1$; $\mathbb{F}(q')$;$\leftarrow q'[m]$; $\mathbb{S}(q') \leftarrow 0$;
      unmarked $\leftarrow$ unmarked $\cup \{q'\}$;

Algorithm 1: Algorithm MA_Build(w) computing, given a string w, the deterministic MA $\mathcal{A}_w$ such that on input x, $d_r(w, x)$ is computed as $\mathcal{A}_w(x)$.

Table 1. Table computing the edit distance between `ab` and `baaaba`

b	2	1	2	2	3	3	4
a	1	1	1	2	3	4	5
λ	0	1	2	3	4	5	6
	λ	b	a	a	a	b	a

Table 2. Table computing d_r(`ab`, `baaaba`)

b	2	0	0	-1	-1	-2	-2
a	1	0	-1	-1	-1	-1	-1
λ	0	0	0	0	0	0	0
	λ	b	a	a	a	b	a

even when exponential, the construction does terminate, and the following result can be given:

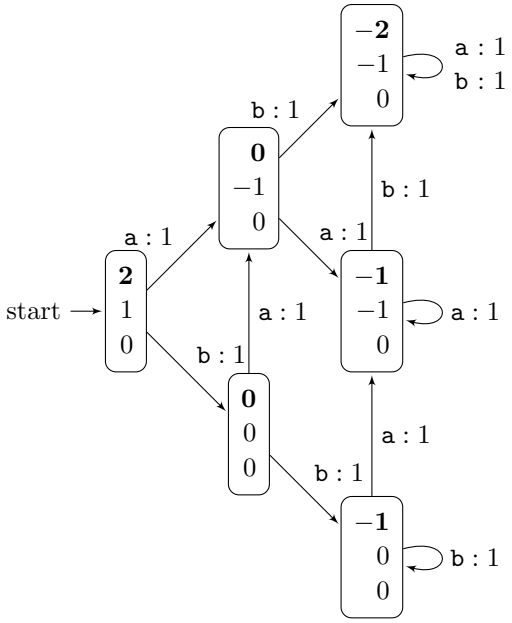

Fig. 1. A first construction. Final state weights are marked in bold.

Proposition 1. *Given any string* u, $d_r(w, u) = \mathcal{A}_w(u)$

To serve as an example, we represent in Fig. 1 the construction for $w = \texttt{ab}$. The following can be noticed:

- this is a finite state machine
- this is a deterministic multiplicity automaton, since there is a single sequence of states when parsing an input string
- given an input string u, $d_e(\texttt{ab}, u)$ can be computed by parsing the string, adding the top value of the vector of the state we finish in to the length of the string. For example,
  - $d_e(\texttt{ab}, \lambda)$=0+2=2
  - $d_e(\texttt{ab}, \texttt{b})$=1+0=1
  - $d_e(\texttt{ab}, \texttt{aaab})$=4-2=2

3.2. *Computing the Product Automaton (Step 2)*

We are now given a PFA $\mathcal{A}_\mathcal{D} = \langle \Sigma, Q_\mathcal{D}, \mathbb{S}_\mathcal{D}, \mathbb{F}_\mathcal{D}, \delta_\mathcal{D} \rangle$ and a multiplicity automaton $\mathcal{A}_w = \langle \Sigma, Q_w, \mathbb{S}_w, \mathbb{F}_w, \delta_w \rangle$.

The product machine, denoted by $\mathcal{A}_{\mathcal{D},w}$ has as states pairs $\langle q, q' \rangle$ with $q \in Q_\mathcal{D}$, $q' \in Q_w$. $\mathcal{A}_{\mathcal{D},w} = \langle \Sigma, Q_{\mathcal{D},w}, \mathbb{S}_{\mathcal{D},w}, \mathbb{F}_{\mathcal{D},w}, \delta_{\mathcal{D},w} \rangle$:

- $\delta_{\mathcal{D},w}(\langle q, q' \rangle, a, \langle s, s' \rangle) = \delta_\mathcal{D}(q, a, s)\delta_w(q', a, s')$,

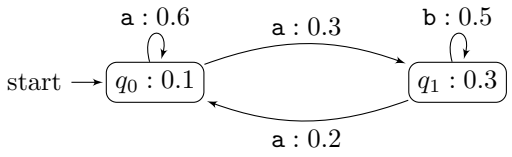

Fig. 2. The PFA for our second construction.

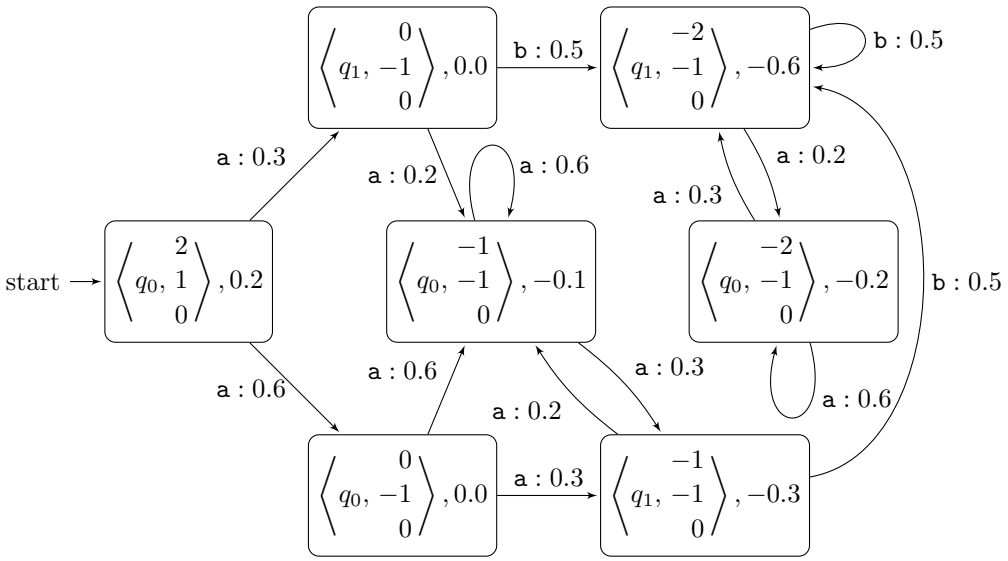

Fig. 3. Building the product automaton

- $\mathbb{S}_{\mathcal{D},w}(\langle q, q' \rangle) = \mathbb{S}_{\mathcal{D}}(q)\, \mathbb{S}_w(q')$,
- $\mathbb{F}_{\mathcal{D},w}(\langle q, q' \rangle) = \mathbb{F}_{\mathcal{D}}(q)\, \mathbb{F}_w(q')$.

By construction, $\mathcal{A}_{\mathcal{D},w}(u) = d_r(w,u) \Pr_{\mathcal{A}}(u)$.

Following the example introduced above, Fig. 3 represents the product automaton for the MA of Fig. 1 ($w = \mathtt{ab}$) and the PFA shown in Fig. 2.

3.3. *Computing the Distance (Step 3)*

We have to compute $d_e(w, \mathcal{A}_{\mathcal{D}}) = \sum_{x \in \Sigma^*} |x| \Pr_{\mathcal{A}}(x) + \sum_{x \in \Sigma^*} d_r(w,x) \Pr_{\mathcal{A}}(x)$.

Let $(\Sigma, \overset{\mathcal{D}}{\mathbf{S}}, \overset{\mathcal{D}}{\mathbf{M}}, \overset{\mathcal{D}}{\mathbf{F}})$ be the matrix representation of the PFA $\mathcal{A}_{\mathcal{D}}$. Since $(I - \overset{\mathcal{D}}{\mathbf{M}})$ is non-singular by definition of PFA, the average length of the strings generated by

$\mathcal{A}_{\mathcal{D}}$ can be computed as in [9]:

$$\sum_{x\in\Sigma^*} |x| \Pr_{\mathcal{A}}(x) = \sum_{i=0}^{\infty} i \Pr_{\mathcal{A}}(\Sigma^i) = \sum_{i=0}^{\infty} i \overset{\mathcal{D}}{\mathbf{S}} \overset{\mathcal{D}}{\mathbf{M}}_{\Sigma}{}^{i} \overset{\mathcal{D}}{\mathbf{F}} = \overset{\mathcal{D}}{\mathbf{S}} \overset{\mathcal{D}}{\mathbf{M}}_{\Sigma} (I - \overset{\mathcal{D}}{\mathbf{M}}_{\Sigma})^{-2} \overset{\mathcal{D}}{\mathbf{F}}$$

Let $(\Sigma, \overset{w}{\mathbf{S}}, \overset{w}{\mathbf{M}}, \overset{w}{\mathbf{F}})$ be the matrix representation of $\mathcal{A}_{D,w}$. Each addend of the series $\sum_{x\in\Sigma^*} d_r(w,x) \Pr_{\mathcal{A}}(x)$ can be computed as:

$$\sum_{x\in\Sigma^*} d_r(w,x) \Pr_{\mathcal{A}}(x) = \sum_{x\in\Sigma^*} \overset{w}{\mathbf{S}} \overset{w}{\mathbf{M}}_x \overset{w}{\mathbf{F}} = \sum_{i=0}^{\infty} \overset{w}{\mathbf{S}} \overset{w}{\mathbf{M}}{}^{i} \overset{w}{\mathbf{F}} = \overset{w}{\mathbf{S}} (I - \overset{w}{\mathbf{M}})^{-1} \overset{w}{\mathbf{F}}$$

One point to check is that the matrix $(I - \overset{w}{\mathbf{M}})$ is non-singular.

By construction, $[\overset{w}{\mathbf{M}}]_{i,j} \geq 0$. Moreover, in any adjacency matrix, $[\mathbf{M}^k]_{i,j}$ is the sum of the weights of all the paths of length exactly k that goes from node i to node j. In our case, by construction, $[\overset{\mathcal{D}}{\mathbf{M}}{}^k]_{i,j} = \sum_{q,s} [\overset{w}{\mathbf{M}}{}^k]_{<i,q>,<j,s>}$ hence $[\overset{\mathcal{D}}{\mathbf{M}}{}^k]_{i,j} \geq [\overset{w}{\mathbf{M}}{}^k]_{<i,q>,<j,s>}$. We also know that $(I - \overset{\mathcal{D}}{\mathbf{M}})$ is non-singular so $\lim_{k\to\infty} [\overset{\mathcal{D}}{\mathbf{M}}{}^k]_{i,j} = 0$. Summarising, we have that, $0 \leq \lim_{k\to\infty} [\overset{w}{\mathbf{M}}{}^k]_{<i,q>,<j,s>} \leq \lim_{k\to\infty} [\overset{\mathcal{D}}{\mathbf{M}}{}^k]_{i,j} = 0$, so $\lim_{k\to\infty} [\overset{w}{\mathbf{M}}{}^k]_{i,j} = 0$ and then, $(I - \overset{w}{\mathbf{M}})$ is non-singular.

Therefore, $d_e(w, \mathcal{A}_{\mathcal{D}}) = \overset{\mathcal{D}}{\mathbf{S}} \overset{\mathcal{D}}{\mathbf{M}}_{\Sigma} (I - \overset{\mathcal{D}}{\mathbf{M}}_{\Sigma})^{-2} \overset{\mathcal{D}}{\mathbf{F}} + \overset{w}{\mathbf{S}} (I - \overset{w}{\mathbf{M}})^{-1} \overset{w}{\mathbf{F}}$. It follows:

Theorem 2. *EDD is decidable and the edit distance between a witness string and a* PFA *with rational weights is rational.*

In our example:

$$\mathbf{S} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{M_a} = \begin{pmatrix} 0 & 0.3 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3 & 0.6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 \\ 0 & 0 & 0 & 0 & 0.6 & 0.3 & 0 \\ 0 & 0 & 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 0.6 \end{pmatrix} \quad \mathbf{M_b} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{F} = \begin{pmatrix} 0.2 \\ 0 \\ 0 \\ -0.6 \\ -0.1 \\ -0.3 \\ -0.2 \end{pmatrix}$$

And then we compute:

$$\mathbf{S}(I - \mathbf{M}_{\Sigma})^{-1}\mathbf{F} \qquad \text{where } \mathbf{M}_{\Sigma} = \mathbf{M}_a + \mathbf{M}_b$$

In our case the result is: -0.99

On the other hand we need to compute:

$$\sum_{x\in\Sigma^*} |x| \Pr_{\mathcal{A}}(x)$$

In this case, the matrix representation of $\mathcal{A}_\mathcal{D}$ (Fig.2) is:

$$\mathbf{S} = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad \mathbf{M}_\mathtt{a} = \begin{pmatrix} 0.6 & 0.3 \\ 0.2 & 0 \end{pmatrix} \quad \mathbf{M}_\mathtt{b} = \begin{pmatrix} 0 & 0 \\ 0 & .5 \end{pmatrix} \quad \mathbf{F} = \begin{pmatrix} 0.1 \\ 0.3 \end{pmatrix}$$

We have to compute:

$$\mathbf{S}\mathbf{M}_\Sigma(I - \mathbf{M}_\Sigma)^{-2}\mathbf{F}$$

In our case, the result of this computation is 4.71. Therefore:

$$d_e(w, \mathcal{A}_\mathcal{D}) = 3.72$$

The construction described here is not polynomially bounded. The final computation is (with arbitrary precision and unit computation time for all arithmetic operations) cubic in the size of the product finite state machine. In turn, the size of this machine essentially depends on the length of the input string.

4. An FPRAS for EDD

As can be seen in the experiments, the method described in Sect. 3 may lead to a combinatorial explosion during the construction of $\mathcal{A}_w$. In this section we propose an FPRAS to approximate the value of $d_e(w, \mathcal{A}_\mathcal{D})$.

Alternatively, the result can be seen as a Probably Approximate Correct (PAC) algorithm [25]. The goal of this framework is to learn (in this case, to compute) a concept for which, with high probability, we obtain a sufficiently good approximation of it.

We are given a PFA $\mathcal{A}_\mathcal{D}$, a string w and two values $\epsilon > 0$, $\delta > 0$.

An FPRAS would be an algorithm which, in time polynomial in $|\mathcal{A}_\mathcal{D}|, |w|, \frac{1}{\epsilon}, \frac{1}{\delta}$ computes a value v such that, with probability at least $1 - \delta$,

$$\left| v - d_e(w, \mathcal{A}_\mathcal{D}) \right| \leq \epsilon$$

Theorem 3. *There exists an* FPRAS *computing the expected distance between a string and a distribution given by a* PFA*.*

Proof. We assume that there exists an Algorithm COMPUTE_BOUNDS which returns two values: L, that is the length at which the generation process of the PFA should be stopped, and N, that denotes the size of the sample to be generated from the PFA. Details of how to compute these values are given below.

These numbers are used by Algorithm BUILD_SAMPLE which with high probability and complexity in $O(NL)$ is going to return a correct sample. The main Algorithm EXPECTED_DISTANCE uses this sample and computes the distance.

The complexity of Algorithm BUILD_SAMPLE is in $O(NL)$. There is a (non null, but lower than $\frac{\delta}{2}$) probability that the number of generated samples is less than N.

□

Algorithm BUILD_SAMPLE($\mathcal{A}_\mathcal{D}$, L, N)
Data: a PFA $\mathcal{A}_\mathcal{D}$
Result: a sample S which, with probability $> 1 - \frac{\delta}{2}$, contains N strings
$S \leftarrow \emptyset$;
for $i : 1 \leq i \leq N$ **do**
Generate a string randomly using $\mathcal{A}_\mathcal{D}$. If the string becomes longer than L, stop and do nothing; otherwise, add the generated string to S.
return S

Algorithm 2: Algorithm BUILD_SAMPLE($\mathcal{A}_\mathcal{D}$)

Algorithm EXPECTED_DISTANCE(w, $\mathcal{A}_\mathcal{D}$, ϵ, δ)
Data: a string w, a PFA $\mathcal{A}_\mathcal{D}$, ϵ, δ
Result: the expected distance between w and $\mathcal{A}_\mathcal{D}$
$\langle N, L\rangle \leftarrow$ COMPUTE_BOUNDS($\mathcal{A}_\mathcal{D}, \epsilon, \delta, w$) ;
$S \leftarrow$ BUILD_SAMPLE($\mathcal{A}_\mathcal{D}$, L, N);
$Res \leftarrow 0$;
for $x \in S$ **do** $Res \leftarrow Res + d_e(w, x)$;
return Res/N

Algorithm 3: Algorithm EXPECTED_DISTANCE(w, $\mathcal{A}_\mathcal{D}$, ϵ, δ)

For Theorem 3 to hold we need to show that the size of the sample one has to draw using the PFA is polynomially bounded. Also, the goal is to have a polynomial limit to the length of the strings without this impacting the quality of the result.

Let us use the following notations: as above w is the string, $\mathcal{A}_\mathcal{D}$ is the PFA. We consider N random strings $X_1, .., X_N$ sampled independent and identically distributed (*iid*) using $\mathcal{A}_\mathcal{D}$, with $Y_i = d_e(w, X_i)$ and Y the random variable $\frac{1}{N}\sum_{i=1}^{i=N} Y_i$.

We call Z the event $|Y - E(Y)| \geq \epsilon$. The goal is therefore to bound the probability of Z.

The first result we will use is the fact that PFA generated distributions are bounded by an exponential function [3]:

Lemma 4. *For any* PFA $\mathcal{A}_\mathcal{D}$ *there exists a constant* $c_\mathcal{A} > 0$ *such that*

$$\Pr_{x\sim\mathcal{A}_\mathcal{D}}[|x| \geq t] \leq e^{-c_\mathcal{A} t} \tag{3}$$

holds for all $t \geq 0$*, unless* $\mathcal{A}_\mathcal{D}$ *only generates one string. In the sequel, we will assume that this is never the case.*

It should be noted in the above lemma the crucial part played by constant $c_\mathcal{A}$. This constant corresponds to the probability of having halted when generating a string from any state of the PFA, divided by the length of these strings. In other words

it can be lower bounded by the smallest halting probability in the PFA divided by the number of states. This lower bound is sufficient to ensure that the inverse of $c_{\mathcal{A}}$ depends polynomially with the size of the PFA. Therefore, the length of strings generated by a PFA always follow a *sub-exponential distribution*.

The second key tool is Hoeffding's inequality: for independent random variables $Y_1, \ldots, Y_N$ taking values between 0 and b, and any value $\epsilon > 0$, let $Y = \frac{1}{N}\sum_{i=1}^{i=N} Y_i$ and then $\Pr[Z] = \Pr[|Y - E(Y)| \geq \epsilon] \leq 2e^{\frac{-2N\epsilon^2}{b^2}}$. It should be noticed that for Hoeffding bounds to hold, the value of each Y_i should be less than b, which is not the case with the edit distance to w.

In order to address $\Pr[Z]$, we split the term in two parts as follows. Let $M = \max|X_i|$ and let L be an arbitrary number (which we will define later explicitly) such that:

$\Pr[Z] = \Pr[Z \cap M \leq L] + \Pr[Z \cap M > L]$.

Then, we bound separately:

- $\Pr[Z \cap M \leq L] \leq \Pr[Z \,|\, M \leq L]$ following the definition of conditional probability, and ignoring $\Pr[M \leq L]$; and
- $\Pr[Z \cap M > L] \leq \Pr[M > L]$.

We use Hoeffding's bound on the first part and a Union Bound on the second part.

- $\Pr[Z \,|\, M \leq L] \leq 2e^{\frac{-2\epsilon^2 N}{(|w|+L)^2}}$ because $0 \leq Y_i \leq |w| + L$ (this is (1)). A tricky point here is that the condition $M \leq L$ should not interfere with the hypothesis (necessary for Hoeffding to hold) that the X_i are sampled *iid*. This is the case here because M is a maximum.
- On the other hand $\Pr[M > L] \leq N \Pr[|X| > L]$. And $N \Pr[|X| > L] \leq Ne^{-c_{\mathcal{A}}L}$ by applying (3).

It follows that:

$$\Pr[Z] \leq 2e^{\frac{-2\epsilon^2 N}{(L+|w|)^2}} + Ne^{-c_{\mathcal{A}}L} \tag{4}$$

In order that $\Pr[Z] < \delta$ in above (4), we need to keep both terms low. Many examples are needed for the Hoeffding bounds to hold, but a large value for N is also a negative factor for the sub-exponential bound: the more examples, the higher the probability of generating a string of length greater than L. However, N and L can be optimised together so that $\Pr[Z] \leq \delta$.

Lemma 5. *If N is such that*

$$\frac{\log 2N - \log(\delta)}{c_{\mathcal{A}}} \leq \epsilon\sqrt{\frac{2N}{2\log 2 - \log(\delta)}} - |w|, \tag{5}$$

then $\Pr[Z] \leq \delta$.

Proof. Starting from the premise that

$$\frac{\log 2N - \log(\delta)}{c_{\mathcal{A}}} \leq \epsilon\sqrt{\frac{2N}{2\log 2 - \log(\delta)}} - |w|,$$

we can find a value L such that

$$\frac{\log 2N - \log(\delta)}{c_{\mathcal{A}}} \leq L \leq \epsilon\sqrt{\frac{2N}{2\log 2 - \log(\delta)}} - |w|. \tag{6}$$

On one hand,

$$\frac{\log 2N - \log(\delta)}{c_{\mathcal{A}}} \leq L \implies Ne^{-c_{\mathcal{A}}L} \leq \frac{\delta}{2}$$

On the other hand,

$$L \leq \epsilon\sqrt{\frac{2N}{2\log 2 - \log(\delta)}} - |w| \implies 2e^{\frac{-2\epsilon^2 N}{(L+|w|)^2}} \leq \frac{\delta}{2}$$

Therefore, given that

$$\Pr[Z] \leq 2e^{\frac{-2\epsilon^2 N}{(L+|w|)^2}} + Ne^{-c_{\mathcal{A}}L}$$

it follows that

$$\Pr[Z] \leq \frac{\delta}{2} + \frac{\delta}{2} \leq \delta. \qquad \square$$

We turn to the following lemma to conclude:

Lemma 6. *There exists a $N_{\delta,\epsilon,c_{\mathcal{A}},|w|}$ fulfilling (5) that is polynomially bounded by $\frac{1}{\delta}, \frac{1}{\epsilon}, c_{\mathcal{A}}$ and $|w|$.*

Proof. We use (5) and write $X = \sqrt{2N}$ (so $\log 2N = 2\log X$):

$$\epsilon\sqrt{\frac{1}{2\log 2 - \log(\delta)}}X - \frac{2}{c_{\mathcal{A}}}\log X + \frac{\log(\delta)}{c_{\mathcal{A}}} - |w| \geq 0$$

Let us notice that $\forall x \geq 1$, $\log(x) < \sqrt{x}$. So we can replace $\log X$ by $\sqrt{X}$ in the above because the term with $\log X$ is negative and we need an X that makes the equation be greater than 0. We then rewrite this with new variable $Y = \sqrt{X}$ as

$$\epsilon\sqrt{\frac{1}{2\log 2 - \log(\delta)}}Y^2 - \frac{2}{c_{\mathcal{A}}}Y + \frac{\log(\delta)}{c_{\mathcal{A}}} - |w| \geq 0,$$

which is a quadratic equation that has two possible values. In this case, we can ignore the solution that leads to a negative N, since it does not make sense in our problem. Developing the positive solution,

$$Y \geq \frac{\frac{1}{c_{\mathcal{A}}} + \sqrt{\frac{1}{c_{\mathcal{A}}^2} - \epsilon\sqrt{\frac{1}{2\log 2 - \log\delta}} \cdot \left(\frac{\log\delta}{c_{\mathcal{A}}} - |w|\right)}}{\epsilon \cdot \sqrt{\frac{1}{2\log 2 - \log\delta}}}$$

We set $K = \epsilon\sqrt{\frac{1}{2\log 2 - \log\delta}}$ and $Y = \sqrt[4]{2N}$:

$$N \geq \frac{1}{2}\left(\frac{1}{Kc_{\mathcal{A}}} + \sqrt{\frac{1}{K^2c_{\mathcal{A}}^2} + \frac{1}{K} \cdot \left(\frac{\log\frac{1}{\delta}}{c_{\mathcal{A}}} + |w|\right)}\right)^4$$

We therefore can choose $N = \left(\frac{1}{Kc_{\mathcal{A}}} + \sqrt{\frac{1}{K^2c_{\mathcal{A}}^2} + \frac{1}{K} \cdot \left(\frac{\log\frac{1}{\delta}}{c_{\mathcal{A}}} + |w|\right)}\right)^4$, with $K = \epsilon\sqrt{\frac{1}{2\log 2 - \log\delta}}$, which is polynomial in $\frac{1}{\delta}$, $\frac{1}{\epsilon}$, $c_{\mathcal{A}}$ and $|w|$.

Using (6) one can also compute a value for L and the pair $\langle N, L\rangle$ can be returned when Algorithm Compute_Bounds is called on parameters $\mathcal{A}$, δ, ϵ and $|w|$. □

It should be added that the above proofs show that a polynomial lower bound for N exists. But the bounds can be tightened by finding directly a pair N and L such that

$$2e^{\frac{-2N\epsilon^2}{\max\{|w|,L\}^2}} \leq \frac{\delta}{2}$$

and

$$N \sum_{w\in\Sigma^{>L}} \Pr_A(w) \leq \frac{\delta}{2}.$$

5. Experiments

As a preliminary evaluation, we ran our Fpras with a fixed value of $\delta = 0.01$ and varying values of ϵ on 100 pairs of Pfa and strings w. In all cases, the difference between the real value and the one computed with the Fpras was always less than ϵ, which confirms that the values computed by Compute_Bounds represent a pessimistic lower bound.

In the series of experiments we want to empirically confirm the time complexity of the algorithms. We showed that the MA-based method grows with the size of the witness string, whereas the Fpras is bounded by N, the number of necessary samples, which is closely related to the expected length of a string from the Pfa. In order to focus these experiments on the most relevant issues, we are using the small Pfa shown in Fig. 4. Parameter $p_f \in (0, 0.9)$ allows us to tune the expected lengths nicely: the lower p_f, the higher the length.

The first experiment examines the time complexity using the method described in Sect. 3. Parameter p_f varies so that the expected lengths of the strings are 6.22, 7.75, 9.71, 12.33, and 16. For this experiment, we generate strings randomly and

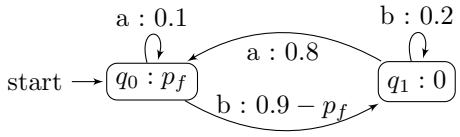

Fig. 4. Parametrizable PFA used in the experiments.

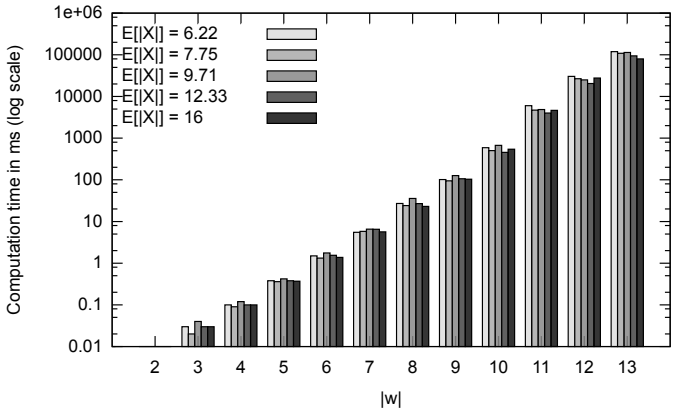

Fig. 5. Average execution time in milliseconds needed for the method based on a MA to compute the distance between a witness string w and a PFA.

uniformly of lengths ranging from 1 to 13 from an alphabet of size 2. Then, we measure the execution time consumed to compute the distance between the string and the PFA, including the construction of the $\mathcal{A}_w$. The experiment is repeated 100 times for each PFA and each witness string length considered. Average results are shown in Fig. 5.

As expected, the complexity of the procedure grows very fast as the length of the witness string is increased. Note that the y-axis is shown in logarithmic scale, so the curve suggests an exponential growth. According to the empirical curve shown in these experiments, computing the distance of a witness string of length 50 would mean an execution time in the order of 10^7 years. Another thing to remark is that this method is not dependent upon the configuration of the PFA, as long as its number of states does not change.

On the other hand, the same experiments are repeated using the FPRAS for ϵ and δ fixed to 0.01. The length of the witness strings are 4, 7, 10, 13 and 16. The parameter p_f is configured so that the expected length of the strings of the PFA are 7.75, 12.33, 16, 21.5, 30.66, 36.31, 44, and 49. Figure 6 illustrates the average results of these experiments.

It can be noticed that the configuration of the PFA is the most relevant factor for the FPRAS. As might be expected by the relationship between the size of the

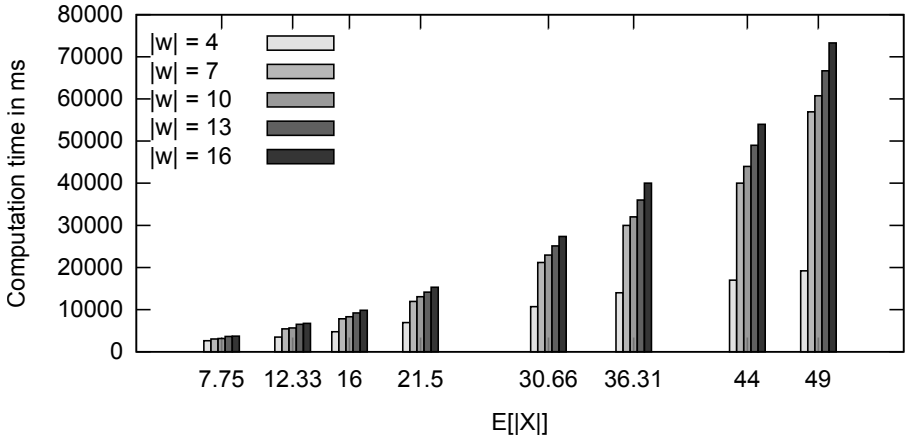

Fig. 6. Average execution time in milliseconds needed for the FPRAS method to compute the distance between a witness string w and a PFA.

PFA measured by $c_{\mathcal{A}}$ and the expected length of the strings, the empirical growth seems to be polynomial [3]. It is also observed that the length of the witness string is a factor that can vary the time complexity since the distances to compute are more expensive. Nevertheless, it is important to emphasize that the FPRAS scales relatively well: for a PFA whose expected length of strings is 100 and a witness string of length 100, the result (89.3235) was computed in around 25 minutes fixing both ϵ and δ to 0.01.

6. Conclusion

Two algorithms have been proposed to deal with the question of computing the expected edit distance from a witness string to a distribution given by a PFA. Whereas one is able to compute this value exactly, it is limited to cases where the length of the witness string is short, as the construction involves building a multiplicity whose size can increase in an exponential way with the length of this string. The first one also shows that the question is decidable and that the solution can be expressed with rational weights.

On the other hand we have a FPRAS which will return with high confidence a value (ϵ)-close to the correct result. It has been shown that its complexity essentially depends on the expected length of the strings of the distribution.

The above results raise several extra questions:

- Computing the expected edit distance between two PFA. In [17] a technique is proposed for the special cases where these PFA correspond to finite languages, or can be determinized. A randomized technique in which strings are drawn from both distributions is likely to work.

- The exact status of EDD remains unclear. The (decision) problem is decidable, as witnessed by Theorem 2. But is it in **NP**?

More importantly, the really crucial question is that of computing the median string, given a PFA.

When given a distribution, a prediction system will often attempt to return the most probable string in order to minimize the empirical risk by following a *maximum a posteriori probability* (MAP) criterion.

Nevertheless, while this may be applicable in a large number of applications, other loss functions can be better suited than the 0/1 loss. For instance, quite often the final goal is to reduce the number of post-processing corrections required to transform a hypothesis into the correct string. This is usually counted by means of the Levenshtein or edit distance (d_e), or a related metric like the Word Error Rate (WER). Then, the empirical risk becomes

$$R(w|x) = \sum_{v \in \Sigma^*} \Pr(v|x) d_e(w, v)$$

In which case, the optimum string is the *median string*. Yet most often the most probable string (or an approximation of it) is proposed instead of the median string, whose search is related to a $\mathcal{NP}$-hard problem [7] even in a finite case. This inconsistency is well known [12], and there have been a number of studies addressing this issue [11, 21], with recently a specific analysis of the relationship between 0/1 loss functions and other discrete loss functions [22]. Other approaches include the introduction of heuristics to approximate the median string [13, 15, 1] in a non-probabilistic scenario.

This constitutes of course a real challenge.

Acknowledgments

The authors wish to acknowledge the help of Borja Balle in establishing the proof of Theorem 3.

This work was partially supported by the Spanish Ministerio de Educación, Cultura y Deporte through a FPU grant (Ref. AP2012-0939) and the Spanish Ministerio de Economía y Competitividad through Project No. TIN2013-48152-C2-1-R (supported by UE FEDER funds).

This work was partly done while the third author was supported by the University of Kyoto.

References

[1] J. Abreu and J.-R. Rico-Juan, A new iterative algorithm for computing a quality approximate median of strings based on edit operations, *Patt. Rec. Letters* **36** (2014) 74–80.

[2] C. Allauzen and M. Mohri, Linear-Space Computation of the Edit-Distance between a String and a Finite Automaton, *CoRR* **abs/0904.4686** (2009).

[3] B. Balle, Learning finite-state machines: Algorithmic and statistical aspects, PhD thesis, Universitat Politécnica de Catalunya (2013).
[4] L. Becerra-Bonache, C. de la Higuera, J. C. Janodet and F. Tantini, Learning balls of strings from edit corrections, *Journal of Machine Learning Research* **9** (2008) 1841–1870.
[5] J. Berstel and C. Reutenauer, *Rational Series and their Languages* (Springer-Verlag, 1988).
[6] J. Calvo-Zaragoza, C. de la Higuera and J. Oncina, Computing the expected edit distance from a string to a PFA, *Proceedings of the 21st International Conference on Implementation and Application of Automata*, (2016), pp. 39–50.
[7] C. de la Higuera and F. Casacuberta, Topology of strings: median string is NP-complete, *Theoretical Computer Science* **230** (2000) 39–48.
[8] C. de la Higuera and J. Oncina, Computing the Most Probable String with a Probabilistic Finite State Machine, *Proceedings of* FSMNLP, (2013).
[9] C. de la Higuera and J. Oncina, The most probable string: an algorithmic study, *Journal of Logic and Computation* **24**(2) (2014) 311–330.
[10] R. Durbin, S. Eddy, A. Krogh and G. Mitchison, *Biological Sequence Analysis: Probalistic Models of Proteins and Nucleic Acids* (Cambridge University Press, Cambridge, UK, 1998).
[11] N. Ehling, R. Zens and H. Ney, Minimum Bayes risk decoding for BLEU, *In Proc. 45th Annual Meeting of the Assoc. for Computational Linguistics (ACL)*, (2007).
[12] F. Jelinek, *Statistical Methods for Speech Recognition* (MIT Press, Cambridge, MA, USA, 1997).
[13] F. Kruzslicz, Improved Greedy Algorithm for Computing Approximate Median Strings, *Acta Cybernetica* **14** (December 1999).
[14] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, *Doklady Akademii Nauk SSSR* **163**(4) (1965) 845–848.
[15] C. D. Martínez-Hinarejos, A. Juan and F. Casacuberta, Median strings for k-nearest neighbour classification, *Patt. Rec. Lett.* **24**(1-3) (2003) 173–181.
[16] S. Mihov and K. U. Schulz, Fast Approximate Search in Large Dictionaries, *Computational Linguistics* **30**(4) (2004) 451–477.
[17] M. Mohri, Edit-Distance Of Weighted Automata: General Definitions and Algorithms, *Int. J. Found. Comput. Sci.* **14**(6) (2003) 957–982.
[18] R. Motwani and P. Raghavan, *Randomized algorithm* (Springer, Berlin, 1995).
[19] G. Navarro and M. Raffinot, *Flexible pattern matching* (Cambridge University Press, Cambridge, UK, 2002).
[20] A. Paz, *Introduction to probabilistic automata* (Academic Press, New York, 1971).
[21] R. Schluter, M. Nussbaum-Thom and H. Ney, On the Relationship Between Bayes Risk and Word Error Rate in ASR, *Audio, Speech, and Language Processing, IEEE Transactions on* **19** (July 2011) 1103–1112.
[22] R. Schluter, M. Nussbaum-Thom and H. Ney, Does the Cost Function Matter in Bayes Decision Rule?, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **34**(2) (2012) 292–301.
[23] K. U. Schulz and S. Mihov, Fast string correction with Levenshtein automata, *IJDAR* **5**(1) (2002) 67–85.
[24] A. Stolcke, Y. Konig and M. Weintraub, Explicit Word Error Minimization in N-Best List Rescoring, *5th. European Conference on Speech Communication and Technology*, (1997).
[25] L. Valiant, A theory of the learnable, *Communications of the ACM* **27**(11) (1984) 1134–1142.

[26] V. Vazirani, *Approximation Algorithms* (Springer, Berlin, 2003).
[27] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta and R. C. Carrasco, Probabilistic Finite State Automata – Part I, *Pattern Analysis and Machine Intelligence* **27**(7) (2005) 1013–1025.
[28] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta and R. C. Carrasco, Probabilistic Finite State Automata – Part I and II, *Pattern Analysis and Machine Intelligence* **27**(7) (2005) 1013–1039.
[29] R. A. Wagner, Order-n correction for regular languages, *Communications of the ACM* **17**(5) (1974) 265—-268.